15장. let, const 키워드와 블록 레벨 스코프

15.1 var 키워드로 선언한 변수의 문제점

15.1.1 변수 중복 선언 허용

1. var 키워드로 선언한 변수는 중복 선언이 가능

2. [예제 15-01]

```
var x = 1;
var y = 1;

// var 키워드로 선언된 변수는 같은 스코프 내에서 중복 선언을 허용.
// 초기화문이 있는 변수 선언문은 자바스크립트 엔진에 의해 var 키워드가

var x = 100;
var y;  // 초기화문이 없는 변수 선언문은 무시된다.

console.log(x);  // 100
console.log(y);  // 1
```

- x변수, y변수는 중복 선언되었다.

15.1.2 함수 레벨 스코프

1. var 키워드로 선언한 변수는 오로지 함수의 코드 블록만을 지역 스코프로 인정

   → 함수 외부에서 var로 선언한 변수는, 코드 블록 내에서 선언해도 모두 전역 변수가 됨.

   = 의도치 않게 변수가 중복 선언되는 경우 발생

15.1.3 변수 호이스팅

1. var로 변수를 선언하면 **변수 호이스팅**에 의해 변수 선언문이 스코프의 선두로 동작한다.

   즉, 변수 호이스팅에 의해 var로 선언한 변수는 변수 선언문 이전에 참조할 수 있다.

   (할당하기 전에 변수를 참조하면 undefined 반환)

15.2 let 키워드

1. var 키워드 단점을 보완하기 위해 ES6에서 **let, const** 도입

15.2.1 변수 중복 선언 금지

1. var 키워드로 동일한 변수를 중복 선언하면 아무런 에러 발생 X

   let 키워드로 동일한 변수를 중복 선언하면 **문법 에러(SyntaxError) 발생**

15.2.2 블록 레벨 스코프

1. var 키워드 : 오로지 함수의 코드 블록만을 지역 스코프로 인정

   let 키워드 : 모든 코드 블록(함수, if문, for문, while문 등)을 지역 스코프로 인정하는

   블록 레벨 스코프(block-level scope)를 따름

15.2.3 변수 호이스팅

1. [예제 15-07]

```
console.log(foo); // ReferenceError(참조 에러) 발생
let foo;
```

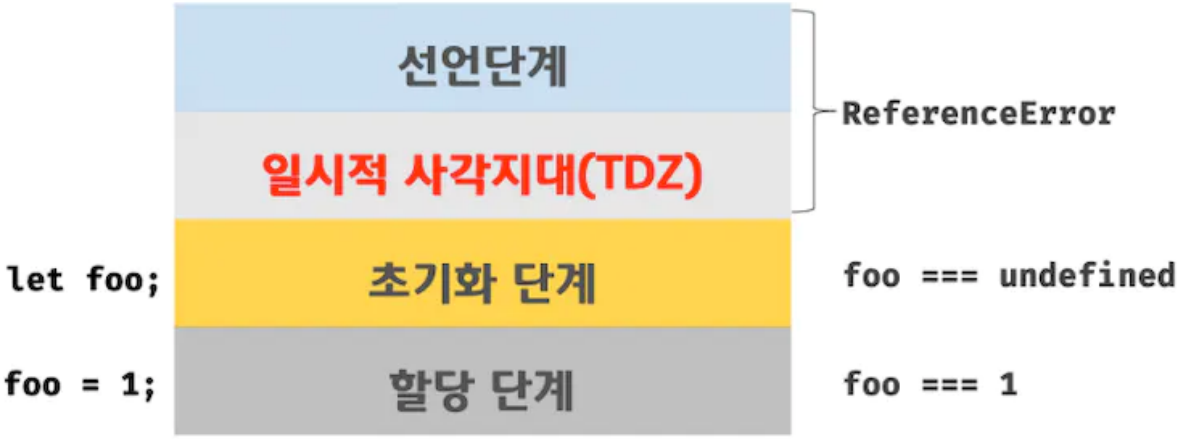
2. var 키워드 : 런타임 이전에 JS 엔진에 의해 암묵적으로 "선언 단계", "초기화 단계"가 한번에 진행

   let 키워드 : **"선언 단계"**, **"초기화 단계"**가 분리되어 진행됨.

⇒ 즉, 런타임 이전에 JS 엔진에 의해 암묵적으로 선언 단계가 먼저 실행되지만,
초기화 단계는 변수 선언문에 도달했을 때 실행됨.

3. **일시적 사각지대(Temporal Dead Zone : TDZ)**

: 스코프의 시작지점 ~ 초기화 시작 지점까지, 변수가 참조할 수 없는 구간



출처 : https://velog.io/@msg_0217/const-let-var로-선언되는-변수의-차이를-설명하시오-딥다이브-15장

결국, let 키워드로 선언한 변수는 호이스팅이 발생하지 않는 것 처럼 보임.

하지만, 그렇지 않다.

4. [예제 15-10]

```
let foo = 1; // 전역 변수

{
    console.log(foo); // ReferenceError 참조 에러
    let foo = 2; // 지역 변수
}
```

자바스크립트는 ES6에서 도입된 let, const를 포함하여,

모든 선언(var, let, const, function, function*, class 등)을 호이스팅한다.

단, ES6에서 도입된 let, const, class를 사용한 선언문은 호이스팅이 발생하지 않는 것처럼 동작

15.2.4 전역 객체와 let

1. [예제 15-11]

```
var x = 1;  // 전역 변수

y = 2;  // 암묵적 전역

function foo(){}  // 전역 함수

console.log(window.x); // 1

console.log(x); // 1

console.log(window.y); // 2

console.log(y); // 2

console.log(window.foo); // f foo(){}

console.log(foo); // f foo(){}
```

let 키워드로 선언한 전역 변수는 전역 객체의 프로퍼티가 아니다.

즉, window.foo와 같이 접근할 수 없다.

15.3 const 키워드

1. const 키워드 : 상수(constant)를 선언하기 위해 사용한다.

15.3.1 선언과 초기화

1. **const 키워드로 선언한 변수는, 반드시 선언과 동시에 초기화해야 한다.**

   그렇지 않으면 문법 에러 발생(SyntaxError)

2. const 키워드 변수도 let과 마찬가지로 블록 레벨 스코프를 가지며,

   변수 호이스팅이 발생하지 않는 것처럼 동작한다.

15.3.2 재할당 금지

1. var, let 으로 선언한 변수는 재할당이 자유로우나 **const로 선언한 변수는 재할당이 금지됨.**

15.3.3 상수

1. const 키워드로 선언한 변수에 원시 값을 할당한 경우 변수 값을 변경할 수 없다.

   (원시 값 = 변경 불가능한 값)

2. 변수의 상대 개념인 **상수는 재할당이 금지된 변수를 말한다.**

3. [예제 15-17]

```
let preTaxPrice = 100; // 세전 가격

let afterTaxPrice = preTaxPrice * (preTaxPrice * 0.1); //
// 0.1의 의미를 명확히 알기 어려워 가독성이 좋지 않음.


console.log(afterTaxPrice); // 110
```

4. **const 키워드로 선언된 변수에 원시 값을 할당한 경우 원시 값은 변경할 수 없는 값이고 const 키워드에 의해 재할당이 금지되므로 할당된 값을 변경할 수 있는 방법은 없다.**

15.3.4 const 키워드와 객체

1. **const 키워드로 선언된 변수에 객체를 할당한 경우, 값을 변경할 수 있다.**

2. [예제 15-19]

```
const person = {
    name : 'Lee'l
};

person.name = 'Kim'; // 객체는 변경 가능한 값이다.

console.log(person); // { name: 'Kim' }
```

const 키워드는 재할당을 금지할 뿐, "불변"을 의미하지는 않는다.

15.4 var vs let vs const

- ES6 사용한다면 var 키워드는 사용 x
- 재할당이 필요한 경우 한정, let 키워드 사용. 이때 변수의 스코프는 최대한 좁게.
- 변경 발생하지 않고 읽기 전용으로 사용(=재할당 필요 없는 상수)하는 원시 값과 객체에는 const

변수를 선언할 때는 일단 **const 키워드**를 사용하자.

반드시 재할당이 필요하다면 그때 let 키워드로 바꿔도 결코 늦지 않다.